One More Try on the FTP

Brian Harvey
SU-AI

Re: File Transfer Protocol May 28, 1975
Ref: RFC 354, 385, 414, 448, 454, 630, 542, 640 1

One More Try on the FTP 2

This is a slight revision of RFC 686, mainly differing in the discussion of print files. Reading several RFCs that I (sigh) never heard of before writing 686 has convinced me that although I was right all along it was for the wrong reasons. The list of reply codes is also slightly different to reflect the four lists in RFCs 354, 454, 542, and 640 more completely. Let me also suggest that if there are no objections before June 1, everyone take it as official that HELP should return 200, that SRVR should be used as discussed below, and that "permanent" 4xx errors be changed to 5xx. And thanks to Jon Postel who just spent all evening helping me straighten this all out. 2a

Aside from a cry of anguish by the site responsible for the security hassle described below, I've only had one comment on this, which was unfavorable but, alas, unspecific. Let me just say, in the hopes of avoiding more such, that I am not just trying to step on toes for the fun of it, and that I don't think the positive changes to FTP-1 proposed here are necessarily the best possible thing. What they are, I think, is easily doable. The great-FTP-in-the-sky isn't showing any signs of universal acceptability, and it shouldn't stand in the way of solving immediate problems. 2b

Leaving Well Enough Alone 3

I recently decided it was time for an overhaul of our FTP user and server programs. This was my first venture into the world of network protocols, and I soon discovered that there was a lot we were doing wrong--and a few things that everyone seemed to be doing differently from each other. When I enquired about this, the response from some quarters was "Oh, you're running Version 1!" 4

Since, as far as I can tell, all but one network host are running version 1, and basically transferring files OK, it seems to me that the existence on paper of an unused protocol should not stand in the way of maintaining the current one unless there is a good reason to

believe that the new one is either imminent or strongly superior or both. (I understand, by the way, that FTP-2 represents a lot of thought and effort by several people who are greater network experts than I, and that it isn't nice of me to propose junking all that work, and I hereby apologize for it.) Let me list what strike me as the main differences in FTP-2 and examine their potential impact on the world. 5

1. FTP-2 uses TELNET-2. The main advantage of the new Telnet protocol is that it allows flexible negotiation about things like echoing. But the communicators in the case of FTP are computer programs, not people, and don't want any echoing anyway. The argument that new hosts might not know about old Telnet seems an unlikely one for quite some time to come; if TELNET-2 ever does really take over the world, FTP-1 could be implemented in it. 5a

2. FTP-2 straightens out the "print file" mess. First of all, there are two separate questions here: what command one ought to give to establish a print file transfer, and which end does what sort of conversion. For the second question, although all of the FTP-1 documents are confusing on the subject, I think it is perfectly obvious what to do: if the user specifies, and the server accepts, an ASCII or EBCDIC print file transfer parameter sequence, then the data sent over the network should contain Fortran control characters. That is, the source file should contain Fortran controls, and should be sent over the net as is, and reformatted if necessary not by the SERVER as the protocol says but by the RECIPIENT (server for STOR, user for RETR). (The "Telnet print file" non-issue will be debunked below.)
As a non-Fortran-user I may be missing something here but I don't think so; it is just like the well-understood TYPE E in which the data is sent in EBCDIC and the recipient can format it for local use as desired. One never reformats a file from ASCII to EBCDIC at the sending end. Perhaps the confusion happened because the protocol authors had in mind using these types to send files directly to a line printer at the server end, and indeed maybe that's all it's good for and nobody's user program will implement TYPE P RETR. 5b

As for the specific commands used to negotiate such a transfer, there may currently be some confusion because the most recent FTP-1 document on the subject (RFC 454) invents a new command, FORM, which is not in general use as far as I know. (Most of my

experiments have been on PDP-10s; perhaps other systems have adopted this command.) FTP-2 puts the format argument in the TYPE command as a second argument. Either way, using a two-dimensional scheme to specify the combinations of ASCII/EBCDIC and ASA/normal conveys no more information than the present A-P-E-F scheme. FTP-2 also introduces the notion of Telnet formatted vs. non-print files. These types are used when a Telnet format oriented system is sending a file to an ASA oriented one, and the recipient needs to know, not what is coming over the net, but how to solve a local file storage problem. It is unnecessary and unfair for hosts to have to negotiate something which does not acttually affect what gets sent over the net. It is unnecessary because the sending user process (there is no problem if the user process is receiving) need not understand what the issue is, it need only make the server understand by transmitting a message from the human user to the server process. Any TYPE parameter must be understood by both processes even if the user treats it just like some other type. 5c

To take a specific example, if I want to send an ASCII file to a 360, my FTP user program needs to have built into it the knowledge that there are two TYPEs which are really the same, AN and AT in the FTP-2 notation. If tomorrow someone needs to know the ultimate use of a binary file (for instance, the old PDP-6 DECtape format stores dump files differently from ordinary data files), I will have to add another piece of information to my FTP user and server (maybe they try to read such a file from me). Instead, information which affects only the RECIPIENT of a file, and not the format AS SENT OVER THE NET, should be specified in some form which the sending process can ignore. This is what the SRVR command should be used for. 5d

If a user at a 360 wants to retrieve a "Telnet print file" from another system, he might tell his FTP user process something like 5e

```
TYPE A
DISP PRINT
RETR FOO etc.
```
5e1

(or whatever syntax they use in their FTP). If a user at a 10 wants to send such a file to a 360, he would say 5f

```
TYPE A
```

SRVR PRINT
STOR FOO etc. 5f1

His FTP user program would send on the SRVR command without comment. Suppose that the transformation is one which might be used in either direction between the same two hosts. (This is not the case for the Telnet print file thing because two 360s would be using ASA format.) Then the user process could accept the equivalent of DISP PRINT from the user, and if the transfer turned out to be a STOR it would decide to send SRVR PRINT first. In this way the FTP user program can be written so that the human user types the same command regardless of the direction of transfer. 5g

Thus, FTP servers which care about the distinction between Telnet print and non-print could implement SRVR N and SRVR T. Ideally the SRVR parameters should be registered with Jon Postel to avoid conflicts, although it is not a disaster if two sites use the same parameter for different things. I suggest that parameters be allowed to be more than one letter, and that an initial letter X be used for really local idiosyncracies. The following should be considered as registered: 5h

T - Telnet print file 5h1

N - Normal. 5h2

Means to turn off any previous SRVR in effect. (This makes "non-print" the default case, rather than making "Telnet print" and "non-print" equal. It is probably a good idea if a user program can count on being able to turn off an earlier SRVR without having to know a specific inverse for it. Servers which do not implement any other SRVR parameters need not implement SRVR N either; user processes shouldn't send SRVR N just for the hell of it.)

3. FTP-2 reshuffles reply codes somewhat. There have been four attempts altogether, that I know of, at specifying a list of reply codes: RFCs 354 and 454 for FTP-1, and RFCs 542 and 640 for FTP-2. There is not much to choose from among the first three of these, which are basically the same, except for a slight increase in specificity each time through, e.g., the introduction of reply

code 456 for a rename which fails because a file of the same (new) name already exists. This increased specificity of reply codes doesn't seem to be much of a virtue; if a rename operation fails, it is the human user, not the FTP user program, who needs to know that it was because of a name conflict rather than some other file system error. I am all for putting such information in the text part of FTP replies. Some real problems are actually addressed in the reply code revision of RFC 640, in which the basic scheme for assigning reply code numbers is more rational than either the FTP-1 scheme or the original FTP-2 scheme. However, I think that most of the benefits of RFC 640 can be obtained in a way which does not require cataclysmic reprogramming. More on this below. 5i

4. FTP-2 was established by a duly constituted ARPAnet committee and we are duty-bound to implement it. I don't suppose anyone would actually put it that baldly, but I've heard things which amounted to that. It's silly. 5j

5. FTP-2 specifies default sockets for the data connection. Most places use the default sockets already anyway, and it is easy enough to ignore the 255 message if you want to. This is a security issue, of course, and I'm afraid that I can't work up much excitement about helping the CIA keep track of what anti-war demonstrations I attended in 1968 and which Vietnamese hamlets to bomb for the greatest strategic effect even if they do pay my salary indirectly. I could rave about this subject for pages, and probably will if I ever get around to writing an argument against MAIL-2, but for now let me just get one anecdote off my chest: I have access to an account at an ARPAnet host because I am responsible at my own site for local maintenance of a program which was written by, and is maintained by, someone at the other site. However, the other site doesn't really trust us outsiders (the account is shared by people in my position at several other hosts) to protect their vital system security, so every week they run a computer program to generate a new random password for the account (last week's was HRHPUK) and notify us all by network mail. Well, on my system and at least one of the others, that mail isn't read protected. I delete my mail when I read it, but since it is hard enough remembering HRHPUK without them changing it every week, I naturally write it in a file on our system. That file could in principle be read protected but it isn't, since sometimes I'm in someone else's office when I want to use

it, and the other passwords in it are for open guest accounts which are widely known. Moral #1: Security freaks are pretty weird. Moral #2: If you have a secret don't keep it on the ARPAnet. (In the past week I have heard about two newly discovered holes in TENEX security.) 5k

6. FTP-2 is available online and FTP-1 isn't, so new hosts can't find out how to do it. Aargh!!! What a reason for doing anything! Surely it would be less costly for someone to type it in again than for everyone to reprogram. Meanwhile these new hosts can ask Jon or Geoff or Bobby or even me for help in getting FTP up. 51

7. FTP-2 has some changes to the strange MODEs and STRUs. This is another thing I can't get too excited about. We support only MODE S and STRU F and that will probably still be true even if we are forced into FTP-2. If the relatively few people who do very large file transfers need to improve the restart capability, they can do so within FTP-1 without impacting the rest of us. The recent implementation of paged file transfers by TENEX shows that problems of individual systems can be solved within the FTP-1 framework. If the IBM people have some problem about record structure in FTP-1, for example, let them solve it in FTP-1, and whatever the solution is, nobody who isn't affected has to reprogram. 5m

Well, to sum up, I am pretty happy with the success I've had transferring files around the network the way things are. When I do run into trouble it's generally because some particular host hasn't implemented some particular feature of FTP-1, and there's no reason to suppose they'll do it any faster if they also have to convert to FTP-2 at the same time. The main thing about FTP-2, as I said at the beginning, is that its existence is an excuse for not solving problems in FTP-1. Some such problems are quite trivial except for the fact that people are reluctant to go against anything in the protocol document, as if the latter were the Holy Writ. A few actually require some coordinated effort. Here is my problem list: 6

1. It is almost true that an FTP user program can understand reply codes by the following simple algorithm: 6a

a. Replies starting with 0 or 1 should be typed out and otherwise ignored. 6a1

b. Replies starting with 2 indicate success (of this step or of the whole operation, depending on the command). 6a2

c. Replies starting with 4 or 5 indicate failure of the command. 6a3

d. Replies starting with 3 are only recognized in three cases: the initial 300 message, the 330 password request, and the 350 MAIL response. (Note that the user program need not distinguish which 300 message it got, merely whether or not it is expecting one right now.) 6a4

The only real problem with this, aside from bugs in a few servers whose maintainers tell me they're working on it, is the HELP command, which is not in the original protocol and which returns 0xx, 1xx, or 2xx depending on the server. (Sometimes more than one message is returned.) The word from one network protocol expert at BBN is that (a) 050 or 030 is the correct response to HELP, and (b) there is a perfectly good mechanism in the protocol for multi-line responses. Unfortunately this does not do much good in dealing with reality. There seems to be a uniform procedure for handling the STAT command: 6b

```
151 information
151 information
151 ...
151 information
200 END OF STATUS                                   6b1
```

which fits right in with the above algorithm. This is despite the fact that 1xx is supposed to constitute a positive response to a command like STAT, so that according to RFC 354 it ought to be 6c

```
151-information
information
...
151 information                                     6c1
```

instead. RFC 414, which approves of the 200 reply for STAT, also gives 200 for HELP. (It seems to me, by the way, that 050 and 030 aren't good enough as responses to HELP since they "constitute neither a positive nor a negative acknowledgement" of

the HELP command and thus don't tell the user program when it ought to ask the human user what to do next.) I suggest that, despite RFC 354, a 200 response be given by all servers at the end of whatever other HELP it gives as of, let's say, June 1. The alternatives are either to let the current rather chaotic situation continue forever while waiting for FTP-2, or to try to standardize everyone on a multi-line 1xx for both HELP and STAT. I'm against changing STAT, which works perfectly for everyone as far as I can tell, and it should be clear that I'm against waiting for FTP-2. Unfortunately there is no real mechanism for "officially" adopting my plan, but I bet if TENEX does it on June 1 the rest of the world will come along. 6d

2. Another reply code problem is the use of 9xx for "experimental" replies not in the protocol. This includes the BBN mail-forwarding message and one other that I know of. This procedure is sanctioned by RFC 385, but it seems like a bad idea to me. For one thing, the user program has no way of knowing whether the reply is positive, negative, or irrelevant. The examples I've been burned by all should have been 0xx messages. I propose that all such messages be given codes in the 000-599 range, chosen to fit the scheme given above for interpreting reply codes. x9x or xx9 could be used to indicate experiments. 6e

3. One more on reply codes: RFC 630 (the one about the TENEX mod to the reply codes for MAIL and MLFL) raises the issue of "temporary" versus "permanent" failures within the 4xx category. RFC 640 deals with this question in the FTP-2 context by changing the meaning of 4xx and 5xx so that the former are for temporary errors and the latter are for permanent errors. I like this idea, and I think it could easily be adapted for FTP-1 use in a way which would allow people to ignore the change and still win. At present, I believe that the only program which attempts to distinguish between temporary and permanent errors is the TENEX mailer. For other programs, no distinction is currently made between 4xx and 5xx responses; both indicate failure, and any retrials are done by the human user based on the text part of the message. A specific set of changes to the reply codes is proposed below. 6f

Perhaps I should make a few more points about RFC 640, since it's the best thing about FTP-2 and the only argument for it I find at

all convincing. Let me try to pick out the virtues of 640 and indicate how they might be achieved in FTP-1. 6g

a. The 3xx category is used uniformly for "positive intermediate replies" where further negotiation in the Telnet connection is required, as for RNFR. I'm afraid this one can't be changed without affecting existing user programs. (One of my goals here is to enable existing user programs to work while some servers continue as now and others adopt the suggestions I make below.) However, although this 3xx idea is logically pleasing, it is not really necessary for a simple-minded user program to be able to interpret replies. The only really new 3xx in RFC 640 is the 350 code for RNFR. But this would only be a real improvement for the user program if there were also a 2xx code which might be returned after RNFR, which is not the case. 640 also abolishes the 300 initial connection message with 220, but again there is clearly no conflict here. 6g1

b. The use of 1xx is expanded to include what is now the 250 code for the beginning of a file transfer. The idea is that a 1xx message doesn't affect the state of the user process, but this is not really true. Consider the file transfer commands. The state diagram on page 13 of RFC 640 is slightly misleading. It appears as if 1xx replies are simply ignored by the user program. In reality, that little loop hides a lot of work: the file transfer itself! If the server replied to the file transfer command immediately with a 2xx message, it would be a bug in the server, not a successful transfer. The real state diagram is more like 6g2

B --> cmd --> W --> 1 --> W --> 2 --> S

(with branches out from the "W"s for bad replies). It should be clear from this diagram that the user program, if it trusts the server to know what it's doing, can expect a 2xx instead of the 1xx without getting confused, since it knows which of the W states it's in. In fact, the use of 1xx in file transfer is very different from its other uses, which are indeed more like the 0xx and 1xx replies in FTP-1. I'd call this particular point a bug in RFC 640. 6g3

c. Automatic programs which use FTP (like mailers) can decide

whether to queue or abandon an unsuccessful transfer based on the distinction between 4xx and 5xx codes. I like this idea, although those temporary errors virtually never happen in real life. This could be accomplished in FTP-1 by moving many of the 4xx replies to 5xx. Mailers would be modified to use the first digit to decide whether or not to retry. This scheme does not cause any catastrophes if some server is slow in converting; it merely leads to unnecessary retries. A few CPU cycles would be wasted in the month following the official switch. Thus, this feature is very different from (a) and (b), which could lead to catastrophic failures if not implemented all at once. (Yes, I know that FTP-2 is supposed to be done on a different ICP socket. I am not discussing FTP-2 but whether its virtues can be transferred to FTP-1.) The specific codes involved are listed below. 6g4

d. The use of the second digit to indicate the type of message. (The proposed division is not totally clean; for example, why is 150 ("file status okay; about to open data connection") considered to be more about the file system than about the data connection?) This can easily be done, since the second digit is not currently important to any user process--the TENEX mailer is, in this plan, already due for modification because of (c). Since this is mostly an aesthetic point, I'm hesitant to do it if it would be difficult for anyone. In particular, I would want to leave the 25x messages alone, in case some user programs distinguish these. This is especially likely for the ones which are entirely meant for the program: 251 and 255. Therefore I propose that if this idea is adopted in FTP-1 the meanings of x2x and x5x be interchanged. This proposal is reflected in the specific list below. 6g5

Let me summarize the specific changes to FTP-1 I'd like to see made, most of which are merely documentation changes to reflect reality: 7

1. HELP should return 200. All commands should return 2xx if successful, and I believe all do except HELP. 7a

2. The definition of 1xx messages should be changed to read: "Informative replies to status inquiries. These constitute neither a positive nor a negative acknowledgment." 7b

3. Experimental reply codes should be of the form x9x or xx9, where the first digit is chosen to reflect the significance of the reply to automated user programs. Reply codes greater than 599 are not permitted. The xx9 form should be used if the reply falls into one of the existing categories for the second digit. User programs are encouraged to determine the significance of the reply from the first digit, rather than requiring a specific reply code, when possible. 7c

4. The STAT command with no argument is considered a request for a directory listing for the current working directory, except that it may be given along with TELNET SYNCH while a transfer is in progress, in which case it is a request for the status of that transfer. (Everyone seems to do the first part of this. I'm not sure if anyone actually implements the second. This is just getting the protocol to agree with reality.) The reply to a STAT command should be zero or more 1xx messages followed by a 200. 7d

5. TYPEs P and F mean that the source file contains ASA control characters and that the recipient program should reformat it if necessary. Servers which care about Telnet-print vs. non-print should implement SRVR T and SRVR N. All user processes should provide a way for the human user to specify an arbitrary SRVR command. 7e

6. (This is just a resolution of a loose end in documentation.) Nested reply codes are not allowed. I don't think this really needs more discussion; they never happen and can't possibly work, and FTP user programs shouldn't have to worry about them. 7f

Here is a list of the current FTP-1 replies, and how they should be renumbered for the new scheme. The changes from 4xx to 5xx should be REQUIRED as of June 1; changes in the second or third digit are not so important. (As explained above, it will not be catastrophic even if some hosts do not meet the requirement.) The list also contains one new possible reply adapted from RFC 640. Replies invented in RFC 454 are so noted; since some of them are for commands largely not implemented like REIN, they may be irrelevant. 7g

OLD	NEW	TEXT	7g1
0x0	0x0	(These messages are not very well defined nor very	

important. Servers should use their judgment.)
100 110 System status reply. (Since nobody does STAT as in
the protocol, this may be a moot point.)
110 111 System busy doing... (This RFC 454 message could easily be considered an example of the one above, but since the 454 authors want to distinguish it, here it is in another number.)
150 150 "File status reply." (If this were really that, it
would be switched to 120, but I believe what is meant
is the response to a bare STAT in mid-transfer, which
is more a connection status reply than a file status
reply.)
151 121 Directory listing reply.
200 200 Last command ok.
201 251 ABOR ok. 7g2
202 252 ABOR ignored, no transfer in progress.
new 206 Command ignored, superfluous here.
230 230 Login complete.
231 231 Logout complete. (RFC 454: Closing connection.)
232 232 Logout command will be processed when transfer is complete. 7g3
233 233 Logout complete, parameters reinitialized. (RFC 454 for REIN) 7g4
250 250 Transfer started correctly.
251 251 MARK yyyy = mmmm
252 252 Transfer completed ok.
253 223 Rename ok.
254 224 Delete ok.
255 255 SOCK nnnn
256 256 Mail completed ok.
300 300 Connection greeting
301 301 Command incomplete (no crlf)
330 330 Enter password 7g5
331 331 Enter account (RFC 454)
350 350 Enter mail. 7g6
400 huh? "This service not implemented." I don't understand
this; how does it differ from 506? If it means no

FTP

		at all, who gave the message? Flush.	7g7
401	451	Service not accepting users now, goodbye.	
430	430	Foo, you are a password hacker!	
431	531	Invalid user or password.	
432	532	User invalid for this service.	
433	533	Need account to write files.	
434	454	Logout by operator.	
435	455	Logout by system.	
436	456	Service shutting down.	
450	520	File not found.	
451	521	Access denied.	
452	452	Transfer incomplete, connection closed.	7g8
453	423	Transfer incomplete, insufficient storage space.	
454	454	Can't connect to your socket.	
455	425	Random file system error (RFC 454)	7g9
456	526	Name duplication, rename failed (RFC 454)	
457	557	Bad transfer parameters (TYPE, BYTE, etc) (RFC 454)	
500	500	Command gibberish.	
501	501	Argument gibberish.	
502	502	Argument missing.	
503	503	Arguments conflict.	
504	504	You can't get there from here.	
505	505	Command conflicts with previous command.	
506	506	Action not implemented.	
507	507	Some other problem. (RFC 454)	
550	520	Bad syntax in pathname. (RFC454)	7g10